Computer Science Department

TECHNICAL REPORT

An Accelerated Bisection Method for the Calculation of Eigenvalues of a Symmetric Tridiagonal Matrix

by

Herbert J. Bernstein*

Technical Report No. 79

July 1983

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

An Accelerated Bisection Method for the Calculation of Eigenvalues of a Symmetric Tridiagonal Matrix

by

Herbert J. Bernstein*

Technical Report No. 79

July 1983

*Permanent address: Chemistry Department, Brookhaven National Laboratory, Upton, New York 11973

An Accelerated Bisection Method for the Calculation of Eigenvalues of a Symmetric Tridiagonal Matrix

Herbert J. Bernstein[‡]

Summary. We present a method for the determination of eigenvalues of a symmetric tridiagonal matrix which combines Givens' Sturm bisection [4,5] with interpolation, to accelerate convergence in high precision cases. By using an appropriate root of the absolute value of the determinant to derive the interpolation weight, results are obtained which compare favorably with the Barth, Martin, Wilkinson algorithm [1].

Subject Classifications: AMS(MOS): 15A18,15-04,65F10; CR: 5.14

1. Introduction

When a modest subset of the eigenvalues of a symmetric tridiagonal matrix is required, the most effective technique available is the bisection method presented by Givens [4,5]. As Wilkinson [6] notes, once an eigenvalue is approximately located, final convergence by interpolation may be more economical than continued bisection. However, in the case of repeated or clustered eigenvalues, interpolation is likely to be more expensive than bisection. Distinguishing the isolated eigenvalues from the repeated ones can often require more code and time than completion of the bisection. Indeed, many obvious techniques for converging to eigenvalues faster than with bisection turn out to take fewer steps, but considerably more time, because so much more has to be done in the inner loop. Further,

[‡] Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. Permanent Address: Chemistry Department, Brookhaven National Laboratory, Upton, New York 11973.

when calculations to near the limiting accuracy of the machine used are required, bisection has the distinct advantage of avoiding convergence questions.

We can retain the guaranteed convergence of bisection by using interpolation only to suggest a new division point, and can solve the problem of repeated eigenvalues by using the appropriate root of the determinant as a weight. This requires us to carry the unscaled determinant value in some form, which is a small change in the orginal bisection technique [4,5]. In those cases where high accuracy is required and the matrix is large this approach provides a significant improvement in computing time as compared to the original bisection technique. However, on machines where high precision division is not very expensive compared to multiplication, the revised bisection technique of Barth, Martin and Wilkinson [1], as further revised by Evans, Shanehchi, and Rick [3], which avoids the cost of repeated rescalings, may compete well with this new approach. On a VAX 11/780 with floating point accelerator, the proposed scheme proved more efficient when looking for eigenvalues to a precision greater than six decimal places, and provided a savings of about 45% at 16 decimal places.

2. Theoretical Background

Let A be a real tridiagonal matrix with major diagonal elements $A_{ii} = \gamma_i$, for i = 1,...,n, and off-diagonal elements $A_{i-1,i} = A_{i,i-1} = \beta_i$, for i = 2,...,n, where n is the order of A. For any υ, one may calculate the determinant of $A-\upsilon$, <u>det</u>$(A-\upsilon)$, by minors from the sequence

(1) $$f_0(\upsilon) = 1,$$

(2) $$f_1(\upsilon) = \gamma_1-\upsilon,$$

(3) $$f_i(\upsilon) = (\gamma_i-\upsilon)f_{i-1}-(\beta_i)^2 f_{i-2}, \qquad i=2,\ldots,n,$$

where <u>det</u>$(A-\upsilon) = f_n(\upsilon)$.

As Givens [5, Theorem 1.5] demonstrated, if we modify this sequence slightly to avoid the case where two consecutive terms are zero, we obtain a count of the eigenvalues of $A \geqslant \upsilon$ as the number of agreements in sign between successive terms, counting zero as positive. It then follows that one can locate any eigenvalue to any required accuracy by successively bisecting an interval known to contain the desired eigenvalue.

Givens' method is numerically very stable. The error in the eigenvalues is no worse than ~15 times the truncation error 2^{-t}, for t-bit mantissa binary machines, independent of the order n of A. However, the direct calculation of the sequence $f_i(\upsilon)$ can easily lead to underflow or overflow, so periodic range checks and rescalings are required. Barth, Martin and Wilkinson [1] avoid the need to rescale by using the sequence

$$(4) \qquad g_i(\upsilon) = f_i(\upsilon)/f_{i-1}(\upsilon), \qquad i=1,\ldots,n,$$

alternatively given as

$$(5) \qquad g_1(\upsilon) = \gamma_1 - \upsilon,$$

$$(6) \qquad g_i(\upsilon) = \gamma_i - \upsilon - (\beta_i)^2/g_{i-1}(\upsilon), \qquad i=2,\ldots,n,$$

with zeros replaced by small non-zero quantities. Counting positive g_i is the same as counting sign agreements of f_i, which yields the count of the number of eigenvalues of $A \geqslant \upsilon$. Equivalently, counting negative g_i is the same as counting sign disagreements of f_i, which yields the count of eigenvalues of $A < \upsilon$. In addition to avoiding rescaling, the use of g_i requires one division in place of two multiplications. Evans, Shanehchi and Rick [3] further improve this scheme by stopping the sequence when no further useful information will be obtained.

As long as undesired eigenvalues are within the test interval, bisection is the most effective technique for approaching the desired eigenvalues. However, once we have an interval containing only one desired eigenvalue, various interpolation schemes can be used to rapidly converge to the eigenvalue. (See Wilkinson [6, p.436 ff.].) For well-isolated eigenvalues, the cost of the additional calculation can often be offset by the reduction in the number of iterations. This is not usually the case when we have repeated eigenvalues or tightly clustered eigenvalues. This problem arises because, while the rate of convergence for a simple eigenvalue found by interpolation is at least quadratic, the rate of convergence for a repeated eigenvalue is usually linear with a coefficient less than the factor of two we get with bisection. (See Wilkinson [6, p.467]). While exact multiple eigenvalues can be avoided by block-decomposition of the tridiagonal matrix, arbitrarily close eigenvalues can arise without the possibility of such decomposition. (e.g. Wilkinson's W^+ matrices [6]). This difficulty can be avoided by using the eigenvalue counts at the interval endpoints to suggest a multiplicity, taking that root of the magnitude of $f_n(\upsilon)$ to linearize the function, interpolating to suggest an optimal division point, and continuing the "bi"-section. We do not suffer from the loss of sign information in taking the root of the absolute value, since we are always doing a strict interpolation with the target assumed to lie within the interval.

To be specific, let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the (necessarily real) eigenvalues of A. Let x_u and x_o be under- and over-estimates of $\lambda_k,\ldots,\lambda_{k+r-1}$, such that

$$(7) \qquad \lambda_{k-1} < x_u \leq \lambda_k \leq\ldots\leq \lambda_{k+r-1} < x_o \leq \lambda_{k+r}.$$

We require a "mean" value λ of $\lambda_k,\ldots\lambda_{k+r-1}$ with the property

$$(8) \qquad (\lambda-x_u)^r(x_o-\lambda_k)\cdot\ldots\cdot(x_o-\lambda_{k+r-1}) = (x_o-\lambda)^r(\lambda_k-x_u)\cdot\ldots\cdot(\lambda_{k+r-1}-x_u).$$

λ exists and lies between λ_k and λ_{k+r-1} by continuity. Now for this mean, λ, we have

$$(9) \qquad \left|\frac{f_n(x_u)}{f_n(x_o)}\right| = \left(\frac{\lambda-x_u}{x_o-\lambda}\right)^r \cdot \left|\Pi\left(1+\frac{\lambda-x_u}{\lambda_j-\lambda}\right)/\Pi\left(1+\frac{\lambda-x_o}{\lambda_j-\lambda}\right)\right|$$

where the products Π run over j not between k and k+r-1. Now suppose that for some ε, $0 < \varepsilon \leq 1/(6n)$, the test interval isolates the eigenvalues within it from the rest by

$$(10) \qquad x_o-x_u \leq \varepsilon|\lambda_i-\lambda_j|, \quad i=k,\ldots,k+r-1,$$

for all j not between k and k+r-1. It then follows from equation (9) that

$$(11) \qquad \left(\frac{\lambda-x_u}{x_o-\lambda}\right)\left(1-4\varepsilon\,\frac{n-r}{r}\right) \leq \left|\frac{f_n(x_u)}{f_n(x_o)}\right|^{1/r} \leq \left(\frac{\lambda-x_u}{x_o-\lambda}\right)\left(1+6\varepsilon\,\frac{n-r}{r}\right),$$

whence it follows that linear interpolation with the rth roots of $|f_n|$ as weights will come within $6n\varepsilon(x_o-x_u)$ of λ. Note that this result depends on our assumption that we have bracketed the desired roots and can do a strict interpolation. Thus in order to continue the process, we must continue to bisect.

Suppose we select a division point by

$$(12)\qquad y_1 = \frac{|f_n(x_u)|^{1/r}\,x_o + |f_n(x_o)|^{1/r}\,x_u}{|f_n(x_u)|^{1/r} + |f_n(x_o)|^{1/r}}.$$

If one of the weights is sufficiently small compared to the other, this division point, y_i, will closely approximate one of the test interval endpoints. This will cause the next step to almost repeat this one, effectively stopping the bisection. This happens when one of the endpoints is very close to an eigenvalue. While this may be satisfactory when we seek an eigenvalue known to be at an extreme of the interval and are satisfied with the estimated error, in general we must find a means to progress. We do this by weighting the weights to draw the new division point away from the endpoints.

Let $\delta < .5$, and

$$(13)\qquad \alpha = \frac{\delta\ |f_n(x_u)|^{1/r} + (1-\delta)\ |f_n(x_o)|^{1/r}}{|f_n(x_u)|^{1/r} + |f_n(x_o)|^{1/r}}.$$

Then

$$(14)\qquad x_1 = \alpha\ x_u + (1-\alpha)x_o$$

lies within the test interval contracted by a factor $1-2\delta$ around the midpoint, and x_1 approaches y_1 as δ approaches zero. We can then continue bisection by raising δ whenever we are trapped by an eigenvalue and accelerate bisection by lowering δ whenever possible.

Now, if δ is non-zero, we will tend to interpolate to a point on the side of the true eigenvalue closer to the interval endpoint which is further from that true eigenvalue. Thus, when δ is sufficiently large, we will tend to alternately bring the high and low endpoints of the test interval closer to the true eigenvalue. For exact calculation, with δ approaching zero, one could hope for nearly quadratic asymptotic convergence of each endpoint, but in practice this cannot be achieved. (See Wilkinson [6, p. 466].)

The alternative technique of reverting to true bisection periodically, as done by Brent [2] in related problems, also converges rapidly, but in test cases tried thus far, does not converge quite as rapidly, since it often places one end-point quite far from the target value. The adaptive changes in δ, above, seem to place both end-points in the vicinity of the target value, while still providing continuation of the bisection as in [2]. Of more value in Brent's algorithm is the use of inverse quadratic interpolation, which seems to save some steps at small expense in some test cases. It is not clear what the proper range of applicability is for large matrices with clustered eigenvalues, so we restrict our attention to linear interpolation.

One might think that a similar, and simpler interpolation could be done using the g_n of equation (4), since this function is already approximately linear near any eigenvalue. This does not work because the range of linearity can be arbitrarily small, as may be seen by calculating g_n for the matrices W_{2m+1}^{+} of Wilkinson [6, p. 308]. We also cannot stop the sequence evaluation early, since we always need the final value of the determinant.

3. Practical Implementation

We apply our interpolation by modifying the bisection algorithm, "bisect", of Barth, Martin and Wilkinson [1]. In the inner loop, we return to Givens' practice [4,5] of computing the determinants instead of ratios of determinants. True zeros are avoided by slight perturbations, as in [1]. We accumulate the rescalings which are done to avoid underflows and overflows as a sum of the logarithms of the scaling factors, which is added to the logarithm of the absolute value of the scaled determinant. Use of the logarithm allows us to avoid further scaling problems and simplifies the necessary taking of roots.

The following FORTRAN subroutine has the same applicability and calling sequence as the Barth, Martin and Wilkinson routine. As with that routine, one would be advised to do the obvious block decompositions of the matrix before calling this new routine, since no internal check is made for zeroes in the off-diagonal terms. This version is for the DEC VAX 11/780 under VMS.

```
        subroutine bisect(gamma,beta,betasq,n,m1,m2,eps1,epsmac,
     * eps2,z,x,estlo)
c
c       Accelerated Sturm Bisection of Tridiagonal Matrix
c       Herbert J. Bernstein, March 1982
c
c       Input formal parameters
c
c       gamma   - the main diagonal of the matrix
c       beta    - the subdiagonal of the matrix, beta(1)=0
c       betasq  - the squares of the subdiagonal
c       n       - the order of the matrix
c       m1,m2   - the range of ordinals of eigenvalues desired
c       eps1    - tolerance of eigenvalues
c       epsmac  - smallest number such that 1+epsmac .gt. 1
c
c       Output formal parameters
c
c       eps2    - attainable tolerance of eigenvalues
c       z       - total number of steps performed
c       x       - resulting eigenvalues
c       estlo   - a scratch array holding lower estimates
c
c       Internal variables
c
c       alpha   - weight of lower endpoint xu in interpolation
c       bdhi    - high bound for rescaling
c       bdhrcp  - reciprocal of bdhi
c       delta   - how closely we may approach ends of interval
c       diff    - difference of logs of roots of determinants
c       epslog  - log of epsmac for bounds on diff
c       f       - previous determinant in calculation by minors
c       fn      - current determinant in calcultation by minors
c       fp      - scratch in going from f to fn
c       fx1     - log of |determinant| at interpolation point x1
c       fxo     - log of |determinant| at high endpoint xo
```

```
c       fxu     - log of |determinant| at low endpoint xu
c       h       - scratch in finding bounds on eigenvalues
c       i       - scratch for loop indices
c       ia      - eigenvalue ordinal bound
c       iexp    - log of scalings done
c       ihi     - count of eigenvalues below xo
c       ii      - scratch for loop indices
c       ilo     - count of eigenvalues below xu
c       k       - ordinal of eigenvalue being found
c       kk      - scratch loop index to compute k
c       loscal  - log of low bound for rescaling
c       relrcp  - reciprocal of epsmac (a big number)
c       width   - width of test interval
c       xl      - interpolation point
c       xlbd    - lower bound for rescaling
c       xlim    - minimum size of test interval
c       xmax    - upper bound for eigenvalues
c       xmin    - lower bound for eigenvalues
c       xnear   - a lower bound on relative distance from xl to xu,xo
c       xo      - upper endpoint of test interval
c       xu      - lower endpoint of test interval
c       xwidth  - next interval width
c
c       Standard functions called
c
c       alog    - natural logarithm
c       amaxl   - maximum value
c       aminl   - minumum value
c       dabs    - real*8 absolute value
c       dexp    - real*8 exponential
c       dfloat  - real*8 from integer
c       float   - real from integer
c       sngl    - real from real*8
c
c       Much of the structure of this routine is derived from the
c       Algol procedure "bisect", by Barth, Martin and Wilkinson,
c       Numer. Math. 9 (1967) 386-393.
c
c       Most non-integer variables must be double precision.
c
        implicit real*8 (a-h,o-z)
c
c       However, the following may be handled at lower precision
c       without serious effect.
c
        real*4 fxl,fxo,fxu,delta,xwidth,xlim,alpha,epslog,diff
        real*4 xnear,relrcp
        integer z
c
c       The arrays are as follows:
c
        dimension gamma(n),beta(n),betasq(n),x(ml:m2),estlo(ml:m2)
c
c       Compute the eigenvalue bounds xmin, xmax
c
        beta(1)=0.
        betasq(1)=0.
        xmin=gamma(n)-dabs(beta(n))
        xmax=gamma(n)+dabs(beta(n))
        do 1000 ii=2,n
        i=n+1-ii
        h=dabs(beta(i))+dabs(beta(i+1))
        if(gamma(i)+h.gt.xmax) xmax=gamma(i)+h
        if(gamma(i)-h.lt.xmin) xmin=gamma(i)-h
 1000   continue
c
c       Compute the tolerances
c
        relrcp=1.d0/epsmac
        epslog=alog(sngl(epsmac))
        if(xmin+xmax.gt.0.)eps2=epsmac*xmax
        if(xmin+xmax.le.0.)eps2=epsmac*(-xmin)
```

```
      if(eps1.le.0.) eps1=eps2
      eps2=0.5d0*eps1+7.d0*eps2
c
c     Derive the parameters for rescaling from epsmac
c
      loscal=epslog/2.
      bdhi=dexp(dfloat(-2*loscal))
      xlbd=dexp(dfloat(loscal))
      bdhrcp=1.d0/bdhi
c
c     inner block
c
      xo=xmax
      do 2000 i=m1,m2
      x(i)=xmax
      estlo(i)=xmin
 2000 continue
      z=0
c
c     Look for the k-th eigenvalue
c
      do 2010 kk=m1,m2
      k=m2+m1-kk
      xu=xmin
c
c     Find initial test interval bounds
c
      do 2020 ii=m1,k
      i=m1+k-ii
      if(xu.ge.estlo(i)) go to 2020
      xu=estlo(i)
      go to 2030
 2020 continue
 2030 if(xo.gt.x(k)) xo=x(k)
c
      fxu=-relrcp
```

```
       fxo=-relrcp
       ilo=0
       ihi=n
       width=relrcp
       xnear=0.
c
c      Loop to divide the interval
c
 3000  xlim=2.d0*epsmac*(dabs(xu)+dabs(xo))+eps1
       xwidth=xo-xu
       if(xwidth.le.xlim) go to 3900
       x1=(xu+xo)/2.d0
       xnear=xnear/2.e0
       if(xwidth*1.333e0.gt.width) xnear=.25e0
       delta=2.e0*xlim/xwidth
       if(delta.gt.2.5e-1) go to 3005
       delta=amax1(delta,xnear)
       diff=amax1(epslog,amin1(-epslog,(fxo-fxu)/float(ihi-ilo)))
       alpha=(1.-delta)*((exp(diff)+delta/(1.-delta))/(exp(diff)+1.))
       x1=dble(alpha)*xu+(1.d0-dble(alpha))*xo
 3005  continue
       width=xwidth
       z=z+1
c
c      Compute determinant by minors
c
       ia=0
       iexp=0
       fn=1.d0
       f=0.d0
c
       do 3100 i=1,n
       fp=f*betasq(i)
       f=fn
       fn=(gamma(i)-x1)*f-fp
       if(fn.lt.0.d0) then
```

```
      ia=ia+1
      fn=-fn
      f=-f
      endif
      do while (fn.le.xlbd)
c
c     Determinant too small, rescale upwards
c
      fn=fn*bdhi
      f=f*bdhi
      iexp=iexp+2*loscal
      if(fn.eq.0.d0) fn=epsmac*f
      enddo
c
c     Determinant too large, rescale downwards
c
      if(fn.gt.bdhi) then
      fn=fn*bdhrcp
      f=f*bdhrcp
      iexp=iexp-2*loscal
      endif
 3100 continue
c
c     Determinant and eigenvalue count found, update parameters
c
      fxl=dmax1(dabs(fn),epsmac)
      fxl=alog(fxl)+float(iexp)
      if(ia.ge.k) go to 3200
c
c     Interpolation point becomes new lower bound
c
      xu=xl
      fxu=fxl
      ilo=ia
      if(ia.lt.ml) estlo(ml)=xl
      if(ia.lt.ml) go to 3150
      estlo(ia+1)=x1
      if(x(ia).gt.x1) x(ia)=x1
 3150 continue
      go to 3000
c
c     Interpolation point becomes new upper bound
c
 3200 xo=x1
      fxo=fx1
      ihi=ia
      go to 3000
 3900 x(k)=(xo+xu)/2.d0
 2010 continue
c
      return
      end
```

4. Organizational Notes and Numerical Properties

The variables eps1, eps2, and epsmac fill the same role as eps1, eps2, and relfeh in Barth, Martin and Wilkinson [1]. We select delta so that the error analysis of [1] applies directly. That is, we always stay far enough from the endpoints that the new division point cannot be confused with the endpoints. In order to force early evaluation near the test interval endpoints we force artificially large negative values of the logarithm of the determinants there. Alternately, we could have stored the values computed earlier, but in practice, the recalculation is a small expense, while storage may be at a premium.

We adjust the δ of equation (9) by testing for a reduction in interval width by a factor of no more than .75. When such a reduction does not occur, we force δ up to .25, and then halve it in each subsequent step. The variable xnear is used to store this information, since δ must also be tested against the eigenvalue tolerance. We could divide xnear by a larger value than two on each step. This further accelerates the convergence in high precision cases, but slows convergence in some pathological cases, mainly at low precision. As long as one is prepared to accept such occasional problems, the divisor of xnear may be raised to, say, ten safely. As the divisor is raised further, the number of possible pathological cases rises, but as Brent [2] notes, they are in any case rare.

An attempt was made to recast this code in a "go-to-free" form. While the numeric properties were, of course, the same, the compiler used produced some serious code inefficiencies due to a poor choice of register bindings. It was felt that publication of the less structured but more efficient code was the lesser of two evils.

5. Test results

The routine above was tested on a variety of matrices on a DEC VAX 11/780. In all cases the eigenvalues produced agreed with those produced by a FORTRAN translation of the Barth, Martin and Wilkinson routine "bisect" to within the prescribed tolerance of eigenvalues. Other variants on [1], such as [3], were tried, and found to actually run slighly slower. The more interesting results follow.

In order to ensure that our use of determinants instead of ratios of determinants would not cause errors due to underflow, the first test matrix of [1] was used. This is a matrix of order 50 with

$$\gamma_1=1,\ \gamma_2=49,\ \beta_2=7,$$

and all other elements zero. There is one eigenvalue of 50 and 49 eigenvalues of zero. When computed with epsl = 10^{-10}, epsmac = 10^{-17}, the accelerated scheme took 39 iterations to produce

$$\lambda_{50}=50.0000000000012,\ \lambda_i=.168706613801504\ 10^{-10},\ i=1,..,49,$$

while the original scheme took 79 iterations to produce

$$\lambda_{50}=49.9999999999773,\ \lambda_i=.227374230554744\ 10^{-10},\ i=1,..,49,$$

with the computed error bound eps2 = .5 10^{-10} in both cases. These results are in reasonably good agreement with the KDF9 results of [1].

The second test matrix of [1],

$$\gamma_i=i^4,\ \beta_i=i-1,\qquad i=1,...,30,$$

was run with epsl = 10^{-8}, 10^{-10}, 10^{-12} and 10^{-14}. The runs took 507, 541, 566 and 577 iterations with the accelerated scheme, versus 1233, 1413, 1567 and 1623 iterations with the original scheme. The results again are in good agreement with those of [1]. For example, the values of λ_1 with the accelerated scheme for decreasing epsl were

.933407087, .93340708487, .9334070848660, .933407084865963,

while those for the original scheme were

.933407086, .93340708482, .9334070848661, .933407084865965,

and those in [1] were, for the higher three values for which the calculation was done,

.933407086, .93340708483, .933407084869,

with all numbers rounded one place beyond the last significant digit to facilitate comparison.

The third test matrix of [1] with

$$\gamma_i=110-10i,\ i=1,\dots,11,\ \gamma_i=10i-110,\ i=12,\dots,21,$$

$$\beta_i=1,\qquad i=2,\dots,21,$$

was run with epsl = 10^{-7}. The accelerated scheme took 195 iterations, while the original scheme took 345 iterations. Similar results were obtained by using various W^+_{2m+1} matrices of Wilkinson [6]. When used with epsl=10^{-8}, the number of iterations from the new scheme were about half those of the original bisection. When used with epsl=10^{-16}, the number of iterations dropped to about 1/2.7 of that for the original scheme. This improvement held in samples of order 21 to 381, despite the clustering of eigenvalues in pairs. Slightly more improvement was seen in testing the W^-_{2m+1} matrices of the same orders, but the results were qualitatively the same.

We now come the question of computer time. The time advantage of this new scheme depends on the characteristics of the computer used. On a machine where a high precision divide is significantly more expensive than two multiplies, the advantage should be seen as soon as the order is large enough for the time in the determinant evaluation to dominate the overhead of the extra tests. However, on the VAX, which has a fast divide, one has to compare the determinant calculation including rescaling with the Barth, Martin, Wilkinson algorithm, which avoids that overhead. In our tests, the advantage showed for almost all matrices when epsl was smaller than than 10^{-6} times the norm of the matrix. This was the the break-even point. At 10^{-16} times the norm of the matrix the new scheme was about 45% faster than the Barth, Martin, Wilkinson algorithm. On the average, the new scheme reduced the interval width by one to two orders of magnitude for each iteration after the interval was below 10^{-6}, and showed approximately quadratic convergence.

For example, for W^{+}_{201} and W^{-}_{201}, for which the norms are $\sim 10^{2}$., the numbers of bisection steps and cpu times in seconds were:

	W^{+}_{201}		W^{-}_{201}	
epsl	BMW steps, time	new steps, time	BMW steps, time	new steps, time
10^{-4}	1489, 6.9	1108, 7.4	2868, 13.1	2116, 14.1
10^{-6}	2236, 10.4	1318, 8.8	4275, 19.8	2476, 16.4
10^{-8}	2995, 13.7	1503, 9.4	5682, 26.3	2769, 18.1
10^{-10}	3650, 16.9	1662, 11.0	6888, 32.5	3029, 20.4
10^{-12}	4421, 19.9	1813, 11.9	8295, 38.0	3282, 22.1
10^{-14}	5139, 23.2	1945, 12.8	9559, 44.5	3477, 23.2
10^{-16}	5549, 24.9	2062, 13.5	10189, 47.3	3603, 23.8

Due to interactive competition, one should expect the times above to vary by up to a second.

Tests were also done on the sensitivity to the divisor of xnear. Values of two, ten, and one hundred were tried on the test cases. The results for dividing by two are above. The other two divisors did not bring the results out of the specified error bounds, and on the average produced similar errors, but in the case of our first test matrix produced eigenvalues which were factors of three and two further from the true values, respectively, than for the smaller divisor. This behavior arises from the fact that we do a final true bisection. With the larger divisors, the second from last interval endpoint was further from the stopping value. One might cure this by simply taking the last endpoint with the lowest determinant, but since we are always within the prescribed error bounds, there is no compelling reason to do so.

In general, the divisor of ten provided a savings for precisions beyond six decimal places, and a slight loss below six decimal places. For example, for W_{201}^{+}, the divisor of ten required 1124 steps at <u>epsl</u>=10^{-4} and 1865 steps at <u>epsl</u>=10^{-16}, versus 1108 steps and 2062 steps for a divisor of two. The divisor of one hundred required 1132 steps and 1870 steps in the same cases. The cpu times changed in proportion.

Acknowledgements

This work was done while on leave at the Courant Institute of Mathematical Sciences. The author wishes to thank the Institute for the facilities provided, Max Goldstein for his helpful suggestions on the code, and Olof Widlund for his detailed comments and for suggesting the Brent reference.

References

1. Barth, W., Martin, R. S., Wilkinson, J. H.: Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method. Numer. Math. 9, 386-393 (1967)
2. Brent, R. P.: Algorithms for minimization without derivatives. Englewood Cliffs, N.J.: Prentice-Hall 1973
3. Evans, D. J., Shanehchi, J., Rick, C. C.: A modified bisection algorithm for the determination of the eigenvalues of a symmetric tridiagonal matrix. Numer. Math. 38, 417-419 (1982)
4. Givens, J. W.: A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices. U.S. Nat. Bur. Standards Applied Mathematics Series 29, 117-122 (1953)
5. Givens, J. W.: Numerical computation of the characteristic values of a real symmetric matrix. Oak Ridge National Laboratory, ORNL-1574 (1954)
6. Wilkinson, J. H.: The Algebraic eigenvalue problem. Oxford: Oxford University Press 1965

*